Formal Analysis of Cognitive Agent Behavior: formal theoretical basis

Sharpanskykh, A.; Treur, J.

2006

document version
Publisher's PDF, also known as Version of record

Formal Analysis of Cognitive Agent Behavior: formal theoretical basis

Technical Report VU-TR-260105

Alexei Sharpanskykh and Jan Treur

Department of Artificial Intelligence, Vrije Universiteit Amsterdam,

De Boelelaan 1081a, NL-1081 HV Amsterdam, The Netherlands

{sharp, treur}@few.vu.nl

In this paper the formal theoretical basis used for transformation of a non-executable external behavioral specification for an agent system into an executable format, required for enabling verification techniques, is explained in detail.

An external behavioral specification for the agent is specified using the Temporal Trace Language (TTL), which syntax and semantics are explained in Section 1.

In the general case an external behavioral agent specification consists of complex temporal relations expressed as dynamic properties in non-causal (and also non-executable) format, which does not allow direct application of automatic verification or simulation. In particular, in order to enable verification and validation of more general dynamic properties against the dynamic properties that constitute the agent behavioral model, an external behavioral description should be replaced by one in a simpler format, closer to a finite state transition system format required, for example, by model checking techniques. In order to obtain this, an agent behavioral description is replaced by one in executable (temporal) format. The justification of such substitution is based on the theorem that a behavioral specification entails a certain dynamic property if and only if the generated executable specification entails the same property. The proof for this theorem and other formal theoretical results are given in Section 2.

Moreover, for the purposes of practical verification by means of model checking techniques, an automated translation from a behavioral specification based on executable temporal logical properties into a finite state transition system description has been developed. The details of a translation procedure are explained in Section 3.

Furthermore, the procedure for translating from the state transition system description into the model specification format for the SMV model checker that is used for verification, as well as the complexity issues of the translated specification are considered in Section 4.

In Sections 5 the application of the proposed approach is illustrated by a paradigmatic example.

1. TTL Syntax and Semantics

The language TTL, short for Temporal Trace Language, is a variant of order-sorted predicate logic. Whereas standard multi-sorted predicate logic is a language to reason about static properties only, TTL is an extension of such languages with facilities for reasoning about the dynamic properties of arbitrary systems expressed by static languages.

1.1 The Language of TTL

For expressing TTL-formulae ontologies are used. In logical terms, an ontology is a **signature** that specifies the vocabulary of a language to represent and reason about a system.

Definition 1.1 (TTL Signature)

A signature consists of the symbols of the following classes:

(1) For representing and reasoning about objects of different ontological kinds a number of syntactical sorts are introduced. Among them are several standard sorts: TIME (a set of all time points), STATE (a set of all state names), TRACE (a set of all trace names; a trace can be considered as a timeline), STATPROP (a set of all state property expressions) and STATOM (a set of all state atoms; STATOM is a subsort of STATPROP). All other sorts, which depend on an application domain, represent subsorts of the general sort OBJECT. Furthermore, for every sort S a sort S_VARS exists, which contains constant names of all possible variables of sort S. A union of all sorts S_VARS for all sorts S constitutes a sort STATE_VARIABLE.

(2) countably infinite number of individual variables of each sort. We shall use t with subscripts and superscripts for variables of the sort TIME; γ with subscripts and superscripts for variables of the sort TRACE; s with subscripts and superscripts for variables of the sort STATE.

(3) a set of constants **C** of each sort, among which true and false of the sort TRUTH_VALUE.

(4) a set of function symbols **Φ**, among which:

a) for each $n \geq 1$, a finite or countably infinite number of function symbols of type $(OBJECT)^n \rightarrow STATPROP$

b) a binary function symbol *comp_aspect* of type ASPECT_COMPONENT x COMPONENT → COMPONENT_STATE_ASPECT, where a sort ASPECT_COMPONENT is a set of the agent aspects (i.e., input, output, and internal); a sort COMPONENT is a set of all agent names; and a sort COMPONENT_STATE_ASPECT is a set of all names of aspects of all agent states.

c) a function symbol *state* of type TRACE x TIME x COMPONENT_STATE_ASPECT → STATE. Sometimes *state* will be used as a binary functional symbol of type TRACE x TIME → STATE.

d) a binary function symbol *truth_value* of type STATE x STATOM → TRUTH_VALUE

e) a binary function symbol $\wedge$ of type STATPROP x STATPROP → STATPROP
- the same for $\vee$ (or), $\rightarrow$ (implication) and $\leftrightarrow$ (equivalence) function symbols

f) a unary function symbol *not* of type STATPROP → STATPROP

g) a binary function symbol $\forall$ (forall) of type STATE_VARIABLE x STATPROP → STATPROP
- the same for the $\exists$ (exists) function symbol

(5) a set of predicate symbols **P,** among which:

a) a predicate symbol *holds* (|=) of type STATE x STATPROP.

b) =: an identity relation on arbitrary sorts

c) <: TIME x TIME is the earlier than relation on time

Definition 1.2 (TTL Signature)

Let L_{TTL}=<OBJECT, STATE_VARIABLE, TIME, STATE, TRACE, STATPROP, **C**, **Φ, P**> be a multi-sorted signature for the TTL language. With each variable x∈ STATE_VARIABLE a sort S is associated, written as x:S. Then the terms and formulas of the language L are defined as follows.

Definition 1.3 (TTL Terms)

The terms of any sort S are inductively defined by:

1. If x:S ∈ STATE_VARIABLE, then x is a term of L_{TTL}
2. If c:S ∈ L_{TTL} is a constant symbol, then c is a term of L_{TTL}
3. If f ∈ L_{TTL} is an n-place function symbol and $\tau_1,\ldots, \tau_n$ are terms of L_{TTL}, then $f(\tau_1,\ldots, \tau_n)$ is a term of L_{TTL}

Definition 1.4 (TTL Formulae)

TTL- formulae are defined inductively as follows:

A. The set of **atomic TTL-formulae** is defined as:

(1) If v_1 is a term of sort STATE, and u_1 is a term of the sort STATPROP, then holds(v_1, u_1) is an atomic TTL formula.

(2) if τ_1, τ_2 are terms of any sort, then = (τ_1, τ_2) is an atomic TTL formula. (further we shall use this predicate in form $\tau_1 = \tau_2$)

(3) if t_1, t_2 are terms of sort TIME, then < (t_1, t_2) is an atomic TTL formula. (further we shall use this predicate in form $t_1 < t_2$, furthermore we shall use $t_1 \leq t_2$ for $t_1<t_2 \wedge t_1=t_2$)

B. The set of **well-formed TTL-formulae** is defined as

(1) Any atomic TTL-formula is a well-formed TTL-formula

(2) If F and G are well-formed TTL-formulae, then so are ~F, (F∨G), (F&G), (F⇒G) and (F⇔G).

(3) If F is a well-formed TTL-formula containing x:S as a free variable, then ($\forall$x:S F(x(S))) and ($\exists$x:S F(x(S))) are well-formed TTL-formulae.

1.2. The Semantics of TTL

An interpretation of a TTL formula is defined by the standard interpretation of order sorted predicate logic formulae.

Definition 1.5 (Interpretation)

An **interpretation** of a TTL formula is defined by a mapping *I* that:

(1) associates each sort symbol S to a certain subdomain D_S, and if $S \subseteq S'$ then $D_S \subseteq Ds'$;

the subdomains for standard sorts is given below:

D_{TRUTH_VALUE}: true, false

D_{TIME}: the set of natural numbers, or the set of real numbers

D_{STATE}: $state_1$, …, $state_n$ (the set of state names)

D_{TRACE}: $trace_1$, …, $trace_n$ (the set of trace names)

D_{STATOM}: the set of state atom names

$D_{STATPROP}$: the set of state property names

$D_{STATE\_VARIABLE}$: the set of state variables names

(2) associates each constant c of sort S to some element of D_s

(3) associates each function symbol f of sort $<X_1, \ldots, X_i> \rightarrow X_{i+1}$ to a mapping $I(X_1)$ x …x $I(X_i) \rightarrow I(X_{i+1})$

(4) associates each predicate symbol P of sort $<X_1, \ldots, X_i>$ to a relation $I(X_1)$ x …x $I(X_i)$

Definition 1.6 (TTL Model)

The **model M** for the language TTL is a pair M=<I, V>, where:

- *I* is an interpretation function, and

- *V* is a variable assignment function, mapping each variable x:S to an element of D_s.

Definition 1.7 (Meaning of TTL terms)

Let M=<I,V> be a model for TTL. Then the meaning of a term $\tau \in$ TTL, denoted by τ^M, is inductively defined by:

1. $(x:S)^M = V(x)$,
2. $(c:S)^M = I(c)$,
3. $f(\tau_1,\ldots,\tau_k)^M = I(f)(\tau_1^M,\ldots, \tau_k^M)$.

Definition 1.8 (Truth definition for TTL)

Let M=<I, V> be a model for TTL. Then the truth definition of TTL is inductively defined by:

1. $\models_M P_i(\tau_1,\ldots,\tau_k)$ iff $I(P_i)$ $(\tau_1^M,\ldots, \tau_k^M) = true$
2. $\models_M \neg\varphi$ iff $\not\models_M \varphi$
3. $\models_M \varphi \wedge \psi$ iff $\models_M \varphi$ and iff $\models_M \psi$
4. $\models_M \forall x:S(\varphi(x))$ iff $\models_{M[x/v]} \varphi(x)$ for all $v \in D_s$

The semantics of connectives and quantifiers is defined in the standard way.

1.3. Axioms of TTL

(1) Substitution function: for all functional symbols $f \in L_{TTL}$ and for all $x:S \in L_{TTL}$, and $x^* \in$ S_VARS $subst_{x^*} f(x) = f(x^*)$

(2) Congruence of traces:

$\forall\gamma_1, \gamma_2 [\forall t [state(\gamma_1, t) = state(\gamma_2, t)] \Rightarrow \gamma_1 = \gamma_2]$

(3) Equality of states:

$\forall s_1, s_2 [\forall a:STATOMS [truth\_value(s_1, a) = truth\_value(s_2, a)] \Rightarrow s_1=s_2]$

(4) Truth value in a state:

$holds(s, p) \Leftrightarrow truth\_value(s, p)=true$

(5) State property semantics

a. $holds(s, (p_1 \wedge p_2)) \Leftrightarrow holds(s, p_1)$ & $holds(s, p_2)$

b. $holds(s, (p_1 \vee p_2)) \Leftrightarrow holds(s, p_1)$ | $holds(s, p_2)$

c. $holds(s, not(p_1)) \Leftrightarrow \neg holds(s, p_1)$

For any constant variable name x* from the sort S_VARS:

d. $holds(s, (\exists x^*: S\_VARS, F(x*))) \Leftrightarrow \exists x:S\ holds(s, F(x))$

e. $holds(s, (\forall x^*: S\_VARS, F(x*))) \Leftrightarrow \forall x:S\ holds(s, F(x))$

(6) Partial order axioms for the sort TIME:

a. $\forall t\ t \leq t$

b. $\forall t_1,t_2 [t_1 \leq t_2 \wedge t_2 \leq t_1] \Rightarrow t_1=t_2$

c. $\forall t_1, t_2, t_3 [t_1 \leq t_2 \wedge t_2 \leq t_3] \Rightarrow t_1 \leq t_3$

2. FORMAL JUSTIFICATION FOR THE TRANSFORMATION PROCEDURE

Lemma 1 (Normalization lemma)
Let t be a given time point. If a formula δ(γ, t) only contains temporal relations such as t' < t" and t' ≤ t", and atoms of the form state(γ, t) |= p for some state formula p, then some state formula q(t) can be constructed such that δ(γ, t) is equivalent to the formula δ*(γ, t) of the form state(γ, t) |= q(t).

Proof sketch for Lemma 1.

First in the formula δ(γ, t) replace all temporal relations such as t' < t" and t' ≤ t" by state(γ, t) |= t' < t" and state(γ, t) |= t' ≤ t" respectively. Then proceed by induction on the composition of the formula δ(γ, t). Treat the logical connectives &, |, ¬, ⇒, ∀s, ∃s.

1) conjunction: δ(γ, t) is δ1(γ, t) & δ2(γ, t)

By induction hypothesis

δ1(γ, t) ⇔ state(γ, t) |= p1 (which is δ1*(γ, t))

δ2(γ, t) ⇔ state(γ, t) |= p2 (which is δ2*(γ, t))

Then

δ(γ, t) ⇔ state(γ, t) |= p1 & state(γ, t) |= p2 ⇔ state(γ, t) |= [p1 ∧ p2] (which becomes δ*(γ, t))

2) disjunction: δ(γ, t) is δ1(γ, t) ∨ δ2(γ, t)

Again by induction hypothesis

δ1(γ, t) ⇔ state(γ, t) |= p1 (which is δ1*(γ, t))

δ2(γ, t) ⇔ state(γ, t) |= p2 (which is δ2*(γ, t))

Then

δ(γ, t) ⇔ state(γ, t) |= p1 | state(γ, t) |= p2 ⇔ state(γ, t) |= [p1 ∨ p2] (which becomes δ*(γ, t))

3) negation: δ(γ, t) is ¬δ1(γ, t)

δ1(γ, t) ⇔ state(γ, t) |= p1

δ(γ, t) ⇔ ¬state(γ, t) |= p1

δ(γ, t) ⇔ state(γ, t) |= not(p1) (which is δ*(γ, t))

4) implication: δ(γ, t) is δ1(γ, t) ⇒ δ2(γ, t)

Again by induction hypothesis

δ1(γ, t) ⇔ state(γ, t) |= p1 (which is δ1*(γ, t))

δ2(γ, t) ⇔ state(γ, t) |= p2 (which is δ2*(γ, t))

Then

δ(γ, t) ⇔ [state(γ, t) |= p1 ⇒ state(γ, t) |= p2] ⇔ state(γ, t) |= [p1 → p2] (which becomes δ*(γ, t))

5) universal quantifier:

δ(γ, t) ⇔ ∀t' state(γ, t) |= p1(t')

δ(γ, t) ⇔ state(γ, t) |= ∀t' p1(t') (which is δ*(γ, t))

6) existential quantifier:

δ(γ, t) ⇔ ∃t' state(γ, t) |= p1(t')

δ(γ, t) ⇔ state(γ, t) |= ∃t' p1(t') (which becomes δ*(γ, t))

Definition 2.1 (Uniqueness and correctness of time)
To relate time within a state property to time external to states a functional symbol present_time is used. Here time is assumed to have the properties of correctness and uniqueness:

Uniqueness of time

This expresses that present_time(t) is true for at most one time point t:

$\forall t, t''\ state(\gamma, t) \models present\_time(t'') \Rightarrow \forall t', t' \neq t''\ \neg state(\gamma, t) \models present\_time(t')$

Correctness of time

This expresses that present_time(t) is true for the current time point t:

$\forall t\ state(\gamma, t) \models present\_time(t)$

Definition 2.2 (Memory formula)
The formula $\varphi_{mem}(\gamma, t)$ obtained by replacing all occurrences in $\varphi_p(\gamma, t)$ of subformulae of the form $state(\gamma, t') \models p$ by $state(\gamma, t) \models memory(t', p)$ is called the *memory formula for* $\varphi_p(\gamma, t)$.

Definition 2.3 (Normalized memory state formula)
The state formula constructed by Lemma 1 for a memory formula $\varphi_{mem}(\gamma, t)$ is called *the normalized memory state formula for* $\varphi_{mem}(\gamma, t)$ and denoted by $q_{mem}(t)$. Moreover, q_{mem} is the state formula $\forall t'[present\_time(t') \rightarrow q_{mem}(t')]$.

Lemma 2
If time has properties of correctness and uniqueness, then

$\varphi_{mem}(\gamma, t) \Leftrightarrow state(\gamma, t) \models q_{mem}(t) \Leftrightarrow state(\gamma, t) \models q_{mem}$ (1)

Proof.

The proof for Lemma 2 follows directly from the Lemma 1, definitions of correctness and uniqueness of time and the definition of the formula q_{mem}. Lemmas 3, 4 and 5 can be proven in the same manner.

Definition 2.4 (Executable theory from interaction to memory)
For a given $\varphi(\gamma, t)$ the executable theory from observation states to memory states $Th_{o\rightarrow m}$ consists of the formulae:
For any atom p occurring in $\varphi_p(\gamma, t)$, expressed in the InteractionOnt(A) for an agent A:

$\forall t'\ state(\gamma, t') \models p \Rightarrow state(\gamma, t') \models memory(t', p)$,

$\forall t''\ state(\gamma, t'') \models memory(t', p) \Rightarrow state(\gamma, t''+1) \models memory(t', p)$,

$state(\gamma, 0) \models present\_time(0)$,

$\forall t\ state(\gamma, t) \models present\_time(t) \Rightarrow state(\gamma, t+1) \models present\_time(t+1)$,

The last two rules are assumed to be included into two following theories $Th_{m\rightarrow p}$ and $Th_{p\rightarrow o}$ as well.

Proposition 1
Let $\varphi_p(\gamma, t)$ be a past statement for a given t, $\varphi_{mem}(\gamma, t)$ the memory formula for $\varphi_p(\gamma, t)$, $q_{mem}(t)$ the normalized memory state formula for $\varphi_{mem}(\gamma, t)$, and $Th_{o\rightarrow m}$ the executable theory from the interaction states for $\varphi_p(\gamma, t)$ to the memory states. Then,

$Th_{o\rightarrow m} \models [\varphi_p(\gamma, t) \Leftrightarrow \varphi_{mem}(\gamma, t)]$

and

$Th_{o\rightarrow m} \models [\varphi_p(\gamma, t) \Leftrightarrow state(\gamma, t) \models q_{mem}(t) \Leftrightarrow state(\gamma, t) \models q_{mem}]$.

Proof.

From the definitions of $q_{mem}(t)$ and of $Th_{o\rightarrow m}$ follows

$Th_{o\rightarrow m} \models [\varphi_p(\gamma, t) \Leftrightarrow \varphi_{mem}(\gamma, t)]$

Further by Lemma 2

$Th_{o\rightarrow m} \models [\varphi_p(\gamma, t) \Leftrightarrow state(\gamma, t) \models q_{mem}(t)]$ ■

Definition 2.5 (Normalized condition state formula)
The state formula constructed by Lemma 1 for $\varphi_{cmem}(\gamma, t, t_1)$ is called *the normalized condition state formula for* $\varphi_{cmem}(\gamma, t, t_1)$ and denoted by $q_{cond}(t, t_1)$. Moreover, $q_{cond}(t)$ is the state formula $\forall t'\ [present\_time(t') \rightarrow q_{cond}(t, t')]$

Lemma 3
If time has properties of correctness and uniqueness, then

$\varphi_{cmem}(\gamma, t, t_1) \Leftrightarrow state(\gamma, t_1) \models q_{cond}(t, t_1) \Leftrightarrow state(\gamma, t_1) \models q_{cond}(t)$ (2)

Proof.

The lemma can be proven in the same manner as Lemma 2.

Definition 2.6 (Normalized preparation state formula)
The state formula constructed by Lemma 1 for $\varphi_{prep}(\gamma, t_1)$ is called *the normalized preparation state formula for* $\varphi_{prep}(\gamma, t_1)$ and denoted by $q_{prep}(t_1)$. Moreover, q_{prep} is the state formula $\forall t' [present\_time(t')] \rightarrow q_{prep}(t')]$

Lemma 4
If time has properties of correctness and uniqueness, then

$\varphi_{prep}(\gamma, t_1) \Leftrightarrow state(\gamma, t_1) \models q_{prep}(t_1) \Leftrightarrow state(\gamma, t_1) \models q_{prep}$ (3)

Proof.

The lemma can be proven in the same manner as Lemma 2.

Definition 2.7 (Conditional preparation formula and normalized conditional preparation state formula)
Let $q_{cond}(t, t_1)$ be the normalized condition state formula for $\varphi_{cmem}(\gamma, t, t1)$ and $q_{prep}(t_1)$ the normalized preparation state formula for $\varphi_{prep}(\gamma, t_1)$. The formula $\varphi_{cprep}(\gamma, t)$ of the form $state(\gamma, t) \models \forall t_1 > t [q_{cond}(t, t_1) \rightarrow q_{prep}(t_1)]$ is called the *conditional preparation formula* for $\varphi_f(\gamma, t)$.

The state formula $\forall t_1 > t [q_{cond}(t, t_1) \rightarrow q_{prep}(t_1)]$ is called *the normalized conditional preparation state formula for* $\varphi_{cprep}(\gamma, t)$ and denoted by $q_{cprep}(t)$. Moreover, q_{cprep} is the formula $\forall t' [present\_time(t') \rightarrow q_{cprep}(t')]$.

Lemma 5
If time has properties of correctness and uniqueness, then

$\varphi_{cprep}(\gamma, t) \Leftrightarrow state(\gamma, t) \models q_{cprep}(t) \Leftrightarrow state(\gamma, t) \models q_{cprep}$ (4)

Proof.

The lemma can be proven in the same manner as Lemma 2.

Definition 2.8 (Executable theory from memory to preparation)
For any state atom p occurring in $\varphi_{cond}(\gamma, t, t_1)$, expressed in the InteractionOnt(A) for the agent A[1]:

$\forall t' \; state(\gamma, t') \models p \Rightarrow state(\gamma, t') \models [memory(t', p) \wedge stimulus\_reaction(p)]$

$\forall t'', t' \; state(\gamma, t'') \models memory(t', p) \Rightarrow state(\gamma, t''+1) \models memory(t', p)$

$\forall t' \; state(\gamma, t') \models q_{mem} \Rightarrow state(\gamma, t') \models q_{cprep}$,

$\forall t', t \; state(\gamma, t') \models [q_{cprep} \wedge q_{cond}(t) \wedge \bigcap_{p} stimulus\_reaction(p)] \Rightarrow state(\gamma, t') \models q_{prep}$,

$\forall t' \; state(\gamma, t') \models [stimulus\_reaction(p) \wedge \neg preparation\_for(t'+c, a)] \Rightarrow state(\gamma, t'+1) \models stimulus\_reaction(p)$,

$\forall t' \; state(\gamma, t') \models [preparation\_for(t'+c, a) \wedge \neg performing\_action(a)] \Rightarrow state(\gamma, t'+c) \models preparation\_for(t'+c, a)$,

where a an action for which $state(\gamma, t'+c) \models performing\_action(a)$ occurs in $\varphi_f(\gamma, t)$.

Proposition 2
Let $\varphi_f(\gamma, t)$ be a future statement for t of the form $\forall t_1 > t [\varphi_{cond}(\gamma, t, t_1) \Rightarrow \varphi_{act}(\gamma, t_1)]$, where $\varphi_{cond}(\gamma, t, t_1)$ is an interval statement, which describes a condition for one or more actions and $\varphi_{act}(\gamma, t_1)$ is a (conjunction of) future statement(s) for t_1, which describes action(s) that are to be performed; let $\varphi_{cprep}(\gamma, t)$ be the conditional preparation formula for $\varphi_f(\gamma, t)$, $q_{cprep}(t)$ be the normalized conditional preparation state formula for $\varphi_{cprep}(\gamma, t)$, and $Th_{m \rightarrow p}$ the executable theory for $\varphi(\gamma, t)$ from memory states to preparation states. Then,

$Th_{m \rightarrow p} \models [\varphi_f(\gamma, t) \Leftrightarrow \varphi_{cprep}(\gamma, t)]$

and

$Th_{m \rightarrow p} \models [\varphi_f(\gamma, t) \Leftrightarrow state(\gamma, t) \models q_{cprep}(t) \Leftrightarrow state(\gamma, t) \models q_{cprep}]$.

Proof.

From the definition of $Th_{m \rightarrow p}$, Lemmas 3 and 4 follows that

$Th_{m \rightarrow p} \models [\varphi_{cond}(\gamma, t, t_1) \Leftrightarrow q_{cond}(t, t_1)]$ (5)

and

$Th_{m \rightarrow p} \models [\varphi_{act}(\gamma, t_1) \Leftrightarrow q_{prep}(\gamma, t_1)]$ (6)

---

[1] If a future formula does not contain a condition, then stimulus_reaction atoms are generated from a corresponding past formula

From (5), (6), definitions of the conditional preparation formula and the normalized conditional preparation state formulae, and the conditions of the proposition it follows

$$Th_{m\to p} \models [\ \varphi_f(\gamma, t) \Leftrightarrow \forall t_1 > t\ [q_{cond}(t, t_1) \to q_{prep}(\gamma, t_1)] \Leftrightarrow \forall t_1 > t\ \varphi_{cprep}(\gamma, t, t_1) \Leftrightarrow \varphi_{cprep}(\gamma, t)\ ]$$

And from Lemma 5 follows

$Th_{m\to p} \models [\ \varphi_f(\gamma, t) \Leftrightarrow \forall t_1 > t\ \ state(\gamma, t) \models q_{cprep}(t, t_1) \Leftrightarrow state(\gamma, t) \models q_{cprep}(t) \Leftrightarrow state(\gamma, t) \models q_{cprep}\ ]$ ■

Proposition 3

Let $\varphi_p(\gamma, t)$ be a past statement for t and $\varphi_f(\gamma, t)$ be a future statement for t. Let $\varphi_{mem}(\gamma, t)$ be the memory formula for $\varphi_p(\gamma, t)$ and $\varphi_{cprep}(\gamma, t)$ the conditional preparation formula for $\varphi_f(\gamma, t)$. Then

$$[\varphi_p(\gamma, t) \Rightarrow \varphi_f(\gamma, t)] \Leftrightarrow [\varphi_{mem}(\gamma, t) \Rightarrow \varphi_{cprep}(\gamma, t)]$$

Proof.

From the Proposition 1 and the Proposition 2 follows

$$\varphi_p(\gamma, t) \Leftrightarrow \varphi_{mem}(\gamma, t) \text{ and } \varphi_f(\gamma, t) \Leftrightarrow \forall t_1 > t\ \varphi_{cprep}(\gamma, t, t_1)$$

Then,

$$[\varphi_p(\gamma, t) \Rightarrow \varphi_f(\gamma, t)] \Leftrightarrow [\varphi_{mem}(\gamma, t) \Rightarrow \forall t_1 > t\ \varphi_{cprep}(\gamma, t, t_1)]$$

So it has been proven that $[\varphi_p(\gamma, t) \Rightarrow \varphi_f(\gamma, t)] \Leftrightarrow [\varphi_{mem}(\gamma, t) \Rightarrow \varphi_{cprep}(\gamma, t)]$ ■

Definition 2.9 (Executable theory from preparation to output)

For a given $\varphi_f(\gamma, t)$ the executable theory from the preparation to the output state(s) $Th_{p\to o}$ consists of the formula

$\forall t'\ state(\gamma, t') \models preparation\_for(t'+c, a) \Rightarrow state(\gamma, t'+c) \models performing\_action(a)$,

where c is a number and a an action for which $state(\gamma, t'+c) \models performing\_action(a)$ occurs in $\varphi_f(\gamma, t)$.

Definition 2.10 (Executable specification)

An *executable specification* $\pi(\gamma, t)$ for the agent A is defined by a union of the dynamic properties from the executable theories $Th_{o\to m}$, $Th_{m\to p}$ and $Th_{p\to o}$.

Definition 2.11 (Coinciding traces)

Two traces γ_1, γ_2 *coincide* on ontology Ont (denoted by $coincide\_on(\gamma_1, \gamma_2, Ont)$) iff

$\forall t\ \forall a \in At(Ont)\quad state(\gamma_1, t) \models a \Leftrightarrow state(\gamma_2, t) \models a$,

where At(Ont) is the set of ground atoms expressed in terms of Ont.

Definition 2.12 (Refinement of an externally observable property)

Let $\varphi(\gamma, t)$ be an externally observable dynamic property for agent A. An executable specification $\pi(\gamma, t)$ for A *refines* $\varphi(\gamma, t)$ iff

(1) $\forall \gamma, t\ \ \pi(\gamma, t) \Rightarrow \varphi(\gamma, t)$

(2) $\forall \gamma_1, t\ [\ \varphi(\gamma_1, t) \Rightarrow [\ \exists \gamma_2\ \ coincide\_on(\gamma_1, \gamma_2, InteractionOnt(A)) \wedge \pi(\gamma_2, t)\ ]\ ]$

Note that for any past interaction statement $\varphi_p(\gamma, t)$ and future interaction statement $\varphi_f(\gamma, t)$ the following holds:

$\forall\ \gamma 1, \gamma 2\ [\ coincide\_on(\gamma 1, \gamma 2, InteractionOnt) \Rightarrow [\varphi_p(\gamma 1, t) \Leftrightarrow \varphi_p(\gamma 2, t) \wedge \varphi_f(\gamma 1, t) \Leftrightarrow \varphi_f(\gamma 2, t)\ ]]$

Lemma 6

Let $\varphi(\gamma, t)$ be a dynamic property expressed using the state ontology Ont. Then the following holds:

(1) $coincide\_on(\gamma_1, \gamma_2, Ont) \wedge coincide\_on(\gamma_2, \gamma_3, Ont) \Rightarrow coincide\_on(\gamma_1, \gamma_3, Ont)$

(2) $coincide\_on(\gamma_1, \gamma_2, Ont) \Rightarrow [\ \varphi(\gamma_1, t) \Leftrightarrow \varphi(\gamma_2, t)\ ]$.

Proof sketch.

The transitivity property (1) follows directly from the definition of coinciding traces for $coincide\_on(\gamma_1, \gamma_2, Ont)$ and $coincide\_on(\gamma_2, \gamma_3, Ont)$:

$\forall a \in At(Ont)\quad \forall t'\quad [state(\gamma_1, t') \models a \Leftrightarrow state(\gamma_3, t') \models a\ ] \Rightarrow coincide\_on(\gamma_1, \gamma_3, Ont)$

From

$\forall\ \gamma_1, \gamma_2$ [coincide_on(γ_1, γ_2, Ont) $\Rightarrow$ $\forall$t' [$\varphi_p(\gamma_1, t') \Leftrightarrow \varphi_p(\gamma_2, t')$ & $\varphi_f(\gamma_1, t') \Leftrightarrow \varphi_f(\gamma_2, t')$]]

follows that $\varphi(\gamma_1, t) \Leftrightarrow \varphi(\gamma_2, t)$.

Note that for any past interaction statement $\varphi_p(\gamma, t)$ and future interaction statement $\varphi_f(\gamma, t)$ the following holds:

$\forall\ \gamma 1, \gamma 2$ [coincide_on($\gamma 1, \gamma 2$, InteractionOnt) $\Rightarrow$ [$\varphi_p(\gamma 1, t) \Leftrightarrow \varphi_p(\gamma 2, t) \wedge \varphi_f(\gamma 1, t) \Leftrightarrow \varphi_f(\gamma 2, t)$]]

Theorem

If the executable specification $\pi_A(\gamma, t)$ refines the external behavioral specification $\varphi_A(\gamma, t)$ of agent A, and $\psi(\gamma, t)$ is a dynamic interaction property of agent A in its environment, expressed using the interaction ontology InteractionOnt(A), then

$$[\forall\gamma\ [\ \pi_A(\gamma, t)\ \Rightarrow\ \psi(\gamma, t)\ ]\ ] \Leftrightarrow\ [\forall\gamma\ [\ \varphi_A(\gamma, t)\ \Rightarrow\ \psi(\gamma, t)\ ]\ ]$$

Proof sketch for Theorem.

$\Leftarrow$ is direct:

from $\pi_i(\gamma, t) \Rightarrow \varphi_i(\gamma, t)$ and $\wedge\varphi_i(\gamma, t) \Rightarrow \psi(\gamma, t)$ it follows $\wedge\pi_i(\gamma, t) \Rightarrow \psi(\gamma, t)$.

$\Rightarrow$ runs as follows:

Suppose $\varphi_i(\gamma, t)$ holds for all i, then since $\pi_1(\gamma)$ refines $\varphi_1(\gamma, t)$, then according to the definition of refinement of an externally observable property exists such a γ_1 that $\pi_1(\gamma_1)$ and coincide_on(γ, γ_1, InteractionOnt (A)).

Due to Lemma 6, this γ_1 still satisfies all $\varphi_i(\gamma_1, t)$ (i.e., $\varphi_i(\gamma_1, t)$ holds for all i).

Proceed with γ_1 to obtain a γ_2 and further for all i to reach a trace γ_n, for which

$\pi_i(\gamma_n)$ holds for all i,

and

coincide_on(γ, γ_n, InteractionOnt(A)),

and

$\varphi_i(\gamma_n)$ holds for all i.

From

$$\forall\gamma\ \forall i\ [\pi_i\ (\gamma)\ \Rightarrow\ \varphi_i\ (\gamma)],$$

and

$$\forall\gamma\ \ [\ \wedge\pi_i(\gamma)\ \Rightarrow\ \psi(\gamma, t)\ ]$$

it follows that $\forall\gamma\ \wedge\varphi_i(\gamma)\ \Rightarrow\ \psi(\gamma)$.

So it has been proven that $\forall\gamma\ \wedge\varphi_i(\gamma)\ \Rightarrow\ \psi(\gamma)$. ■

3. TRANSFORMATION INTO THE FINITE STATE TRANSITION SYSTEM FORMAT

According to the Definition 2.10 the executable specification of an agent's behavior consists of the union of three theories $Th_{o\to m}$, $Th_{m\to p}$ and $Th_{p\to o}$, which in turn contain a number of executable dynamic properties. These dynamic properties can be translated into transition rules for a finite state transition system, based on which the same traces are generated as by executing the dynamic properties. For this purpose we use the predicate present_time(t) introduced earlier, which is only true in a state for the current time point t. Further the executable properties from the executable specification, translated into the transition rules are given.

Time increment rules:

present_time(0) $\wedge\ \neg$p $\rightarrow$ present_time(1)

present_time(t) $\wedge\ \neg q_{mem} \wedge\ \neg$p $\rightarrow$ present_time(t+1)

present_time(t) $\wedge\ q_{cprep} \wedge\ \neg q_{cond}$(t) $\wedge\ \neg$p $\rightarrow$ present_time(t+1)

present_time(t) $\wedge\ q_{prep} \rightarrow$ present_time(t+1)

Memory state creation rule:

For any state atom p occurring in $\varphi_{cond}(\gamma, t, t_1)$, expressed in the InteractionOnt(A) for agent A:

present_time(t) $\wedge$ p $\rightarrow$ [memory(t, p) $\wedge$ stimulus_reaction(p)]

For all other state atoms p

present_time(t) $\wedge$ p $\rightarrow$ memory(t, p)

Memory persistence rule:

memory(t, p) $\rightarrow$ memory(t, p)

Conditional preparation generation rule:

$q_{mem} \rightarrow$ conditional_preparation_for(a),

where a an action for which state(γ, t'+c) |= performing_action(a) occurs in $\varphi_f(\gamma, t)$.

Preparation state creation rule:

present_time(t') $\wedge$ conditional_preparation_for(a) $\wedge$ $q_{cond}(t) \wedge \bigcap_{p}$ stimulus_reaction(p) $\rightarrow$ preparation_for(t'+c, a)

for every subformula of the form

present_time(t') $\rightarrow$ preparation_for(t'+c, a)

that occurs in q_{cprep}.

Preparation state persistence rule:

preparation_for(t+c, a) $\wedge \neg$performing_action(a) $\rightarrow$ preparation_for(t+c, a)

Stimulus reaction state persistence rule:

present_time(t') $\wedge$ stimulus_reaction(p) $\wedge \neg$ preparation_for(t'+c, a) $\rightarrow$ stimulus_reaction(p)

Output state creation rule:

preparation_for(t+c, a) $\wedge$ present_time(t+c-1) $\rightarrow$ performing_action(a), where a is an action.

4. TRANSFORMATION INTO THE SMV MODEL SPECIFICATION FORMAT

For automatic verification of general properties of an agent in its environment by means of model checking techniques, a corresponding to the external agent behavioral specification representation of a finite state transition system should be translated into the input format of one of the existing model checkers. The model checker SMV has been chosen as a verification tool for two reasons. First, the input language of SMV is syntactically and semantically similar to the general description of a finite state transition system, which facilitates automatic translation into the SMV input format. Second, SMV uses efficient symbolic algorithms to traverse a model and the expressive temporal logic CTL for specifying properties to check.

A procedure for translating from a general representation for a finite state transition system into the model specification format for the SMV model checker has been developed. In Section 4.1 the main steps of the procedure are explained. Then, in Section 4.2 the complexity issues of a specification, generated by application of this procedure, are considered.

4.1 Transformation procedure

Let us describe the transformation procedure, which is automatically performed by dedicated software that has been developed.

First, using the standard rules [1] $q_{mem}(t)$ and $q_{cond}(t)$ expressions for each dynamic property DP_n are transformed into the prenex normal form. Then, for each dynamic property the described below steps 1-3 are applied first to $q_{mem}(t)$ and then to $q_{cond}(t)$. After that conditional preparation generation rules are added by performing the step 4. Finally, the preparation and action state creation rules are generated for each dynamic property by performing the step 5.

Step 1. For each occurrence of an existential quantifier of the form $\exists$t1 P(t1), where t1 is a time variable name and P(t1) is some predicate of the form memory(observed(t1, obs_event)) or $\neg$memory(observed(t1, obs_event)), obs_event is some atom, and for each occurrence of a universal quantifier of the form $\forall$t1 P(t1), create an atom (a label) t1 and add to the specification the following:

```
t1: boolean ;
init(t1):=0;
obs_event: boolean;
init(obs_event):=0;
```

Step 2. For each existentially quantified time variable and universally quantified time variable that is not in the scope of any existential quantifier with a time variable:

(a) For each occurrence of the expression Q t1, t2 R t1 memory(observed(t1, obs_event)), where Q is either an existential or a universal quantifier, R is the comparison relation for the linear ordered time line: R={<, ≤}; t1 and t2 are time variables, add to the specification the following rules:

```
  next(t1):= case
              t2 & obs_event: 1; //memory state creation
              !t2: 0;
              1: t1;             //persistence of memory
  esac;
```

(b) For each occurrence of the expression Q t1, t2 R t1 ¬memory(observed(t1, obs_event)), add to the specification the following rules:

```
next(t1):= case
           t2 & !obs_event: 1;
           !t2: 0;
           1: t1;
esac;
```

Step 3. For each expression of the form ∃t1, t2 ∀t3 [t3 R t2 ∧ t1 R t3 & memory(observed(t1, obs_event1)) ∧ memory(observed(t2, obs_event2)) ∧ P3(t3)]:

(a) if P3(t) is of the form memory(observed(t3, obs_event))

i. For t3 < t2 and t1< t3 add to the specification the following rules:

```
t3t1_eq: boolean ;
init(t3t1_eq):=0;
next(t3t1_eq):= case
            t1: 1;
             1: 0;
esac;
next(t1):= case
            !obs_event2 & !t2 & t3t1_eq & !obs_event3: 0;
            1: t1;
esac;
next(t3):= case
           !t1: 0;
           !obs_event2 & !t2 & !obs_event3: 0;
           !obs_event2 & !t2 & obs_event3: 1;
           1: t3;
esac;
```

ii. For t3 < t2 and t1≤ t3 add to the specification the following rules:

```
t3t1_eq: boolean ;
init(t3t1_eq):=0;
next(t3t1_eq):= case
            t1: 1;
             1: 0;
esac;
next(t1):= case
            !t2 & t3t1_eq & !obs_event3: 0;
            1: t1;
esac;
next(t3):= case
           !t1: 0;
           !t2 & !obs_event3: 0;
           !t2 & obs_event3: 1;
           1: t3;
esac;
```

iii. For t3 ≤ t2 and t1< t3 add to the specification the following rules:

```
next(t1):= case
            !obs_event2 & !t2 & !obs_event3: 0;
```

```
            1: t1;
esac;
next(t3):= case
           !t1: 0;
           !obs_event2 & !t2 & !obs_event3: 0;
           !obs_event2 & !t2 & obs_event3: 1;
           1: t3;
esac;
```

iiii. For $t3 \leq t2$ and $t1 \leq t3$ add to the specification the following rules:

```
next(t1):= case
            !t2 & !obs_event3: 0;
            1: t1;
esac;
next(t3):= case
           !t1: 0;
           !t2 & !obs_event3: 0;
           !t2 & obs_event3: 1;
           1: t3;
esac;
```

(b) If $P3(t)$ is of the form $\neg$memory(observed(t3, obs_event))

i. For $t3 < t2$ and $t1 < t3$ add to the specification the following rules:

```
neg_t3t1_eq: boolean ;
init(neg_t3t1_eq):=0;
next(neg_t3t1_eq):= case
            t1: 1;
             1: 0;
esac;
next(t1):= case
         !obs_event2 & !t2 & neg_t3t1_eq & obs_event3: 0;
         1: t1;
esac;
next(t3):= case
           !t1: 0;
           !obs_event2 & !t2 & obs_event3: 0;
           !obs_event2 & !t2 & !obs_event3: 1;
           1: t3;
esac;
```

ii. For $t3 < t2$ and $t1 \leq t3$ add to the specification the following rules:

```
neg_t3t1_eq: boolean ;
init(neg_t3t1_eq):=0;
next(neg_t3t1_eq):= case
            t1: 1;
             1: 0;
esac;
next(t1):= case
            !t2 & neg_t3t1_eq & obs_event3: 0;
            1: t1;
esac;
next(t3):= case
           !t1: 0;
           !t2 & obs_event3: 0;
           !t2 &!obs_event3: 1;
           1: t3;
esac;
```

iii. For $t3 \leq t2$ and $t1 < t3$ add to the specification the following rules:

```
next(t1):= case
            !obs_event2 & !t2 & obs_event3: 0;
            1: t1;
esac;
next(t3):= case
           !t1: 0;
           !obs_event2 & !t2 & obs_event3: 0;
           !obs_event2 & !t2 & !obs_event3: 1;
           1: t3;
esac;
```

iiii. For $t3 \leq t2$ and $t1 \leq t3$ add to the specification the following rules:

```
next(t1):= case
            !t2 & obs_event3: 0;
            1: t1;
esac;
next(t3):= case
           !t1: 0;
           !t2 & obs_event3: 0;
           !t2 &!obs_event3: 1;
           1: t3;
esac;
```

Step 4. Add conditional preparation generation rules to the specification:

```
next(fmemN):= case   // N is a number of a dynamic property in the input specification
            ∧ti: 1;  // conjunction of all labels, created based on φp(γ, t)
            i
            1: 0;
esac;
```

Step 5. For each action predicate performing_action(act) in a formula $q_{act}(t)$ add to the specification the following rules:

```
next(fprep_act):= case
            fmemN & ∧tj: 1;  //conjunction of all labels, created based on φcond(γ, t)
                    j
                      1: 0;
esac;

next(act):= case
               fprep_act: 1;
                       1: 0;
esac;
```

4.2 Complexity issues

The complexity of the representation of the obtained executable model is linear in size of the non-executable behavioral specification. More specifically, the non-executable specification is related to the SMV specification in the following linear way:

(1) for every quantified variable from a non-executable specification a variable and an appropriate rule for its update are introduced;

(2) for every nested quantifier an additional variable and an auxiliary executable rule are introduced, which establishes a relation between the quantified variables;

(3) for every observed atom from a past and a conditional formulae from dynamic properties, a corresponding memory state creation and a memory state persistence rule are introduced using the variables described in (1) and (2), and variables that correspond to external events;

(4) for every non-executable dynamic property auxiliary variables fmem and fprep (i.e., the variables that indicate truth values of $\varphi_{mem}(\gamma, t)$ and $\varphi_{prep}(\gamma, t1)$ respectively) and corresponding update rules are introduced;

(5) for every action specified in $\varphi_{act}(\gamma, t1)$ a variable and an appropriate update rule are introduced;

(6) for reactivation of agent preparation states the auxiliary variables and the update rules corresponding to observed atoms from $\varphi_{prep}(\gamma, t1)$ are introduced.

For verifying an executable model in the SMV OBDD-based symbolic model checking algorithms are used; the study of complexity of such algorithms is given in [2].

5. EXAMPLE

We illustrate the proposed approach by an example from the studies of animal behavior. In this example we investigate two cases of a laboratory mouse behavior (i.e., delayed-response and motivation-based behavior) in two different experimental settings. By means of the approach introduced in this paper we are aiming at determining what type of behavior will bring the agent to the satisfaction state in both environmental settings.

The initial situation is identical for both experimental settings and defined as follows: the mouse is placed in front of a transparent screen that separates it from a piece of food that is put at the position p1 behind the screen. The other possible position for placing food is p2. The mouse is able to observe the position of food and of the screen. In the first experimental setting at some moment after food has been put, a cup is placed covering the food at position p1, which makes food invisible for the mouse. After some time the screen is raised and the animal is free to go to any position. If the mouse comes to the position, where the food is hidden, then it will be capable to lift up the cup and get the food.

According to the proposed approach, first for each experimental setting an external behavioral specification is formally defined. The specification consists of environmental properties and externally observable behavioral properties of the agent. The formal behavioral specifications for the experiments are given below.

Environmental properties:

EP1: At some time point food has been put at the position p1, after some time a cup has been placed upon food and after that the screen is raised

$\exists t1, t2, t3\ t2>t1\ \&\ t2<t3\ state(\gamma, t2) \models cup\_at(p1)\ \&\ state(\gamma, t1) \models food\_at(p1)\ \&\ state(\gamma, t3) \models not\_screen$

EP2: Food stays at the position where it has been put until it has been taken away or the agent is satisfied

$\forall t4\ state(\gamma, t4) \models [\ food\_at(X)\ \&\ not(mouse\_sat)\ \&\ not(food\_taken\_away\_from(X))\ ] \Rightarrow state(\gamma, t4+1) \models food\_at(X)$, where $X \in \{p1, p2\}$

EP3: After the screen has been raised, it will never be drawn down again

$\forall t5\ state(\gamma, t5) \models not\_screen \Rightarrow state(\gamma, t5+1) \models not\_screen$

EP4: After placing the cup it will not be removed

$\forall t6\ state(\gamma, t6) \models cup\_at(X) \Rightarrow state(\gamma, t6+1) \models cup\_at(X)$, where $X \in \{p1, p2\}$

Properties that define the externally observable behavior of the mouse:

BP1: The mouse is able to observe presence (absence) of screen.

$\forall t7\ state(\gamma, t7) \models X \Rightarrow \exists t8\ t8>t7\ state(\gamma, t8, input(mouse)) \models observed(X)$, where $X \in \{not\_screen, screen\}$

BP2: The mouse is always able to observe presence or absence of food if the cup is not covering it.

$\forall t9\ state(\gamma, t9) \models X\ \&\ not(cup\_at(Y)) \Rightarrow \exists t10\ t10>t9\ state(\gamma, t10, input(mouse)) \models observed(X)$, where $X \in \{food\_at(Y), not(food\_at(Y))\}$ and $Y \in \{p1, p2\}$

BP3: The mouse is able to observe that food is taken away if the cup is not covering it.

$\forall t11\ state(\gamma, t11) \models food\_taken\_away\_from(X)\ \&\ not(cup\_at(X)) \Rightarrow \exists t12\ t12>t11\ state(\gamma, t12, input(mouse)) \models observed(food\_taken\_away\_from(X))$, where $X \in \{p1, p2\}$

BP4: The mouse always arrives at the position where it goes.

$\forall t13\ state(\gamma, t13, output(mouse)) \models performing\_action(goto(X)) \Rightarrow \exists t14\ t14>t13\ state(\gamma, t14) \models mouse\_at(X)$, where $X \in \{p1, p2\}$

BP5: If the mouse is at the position with food, then it will be eventually satisfied (after consuming food).

$\forall t15\ state(\gamma, t15) \models mouse\_at(X)\ \&\ food\_at(X) \Rightarrow \exists t16\ t16>t15\ state(\gamma, t16) \models mouse\_sat$, where $X \in \{p1, p2\}$

BP6: The mouse consumes food completely.

$\forall$t17 state(γ, t17) |= mouse_sat & mouse_at(X) $\Rightarrow$ state(γ, t17+1) |= not(food_at(X))

BP7: If mouse found the position with food, it stays there.

$\forall$t18 state(γ, t18) |= mouse_at(X) & food_at(X) $\Rightarrow$ $\forall$t19 t19>t18 mouse_at(X)

BP8: Delayed-response behavior of the mouse

The mouse goes to the position with food if and only if it observes that there is no screen and at some point in the past the mouse observed food and since then did not observe the absence of food.

$\forall$t20 [state(γ, t20, input(mouse)) |= observed(not_screen) & $\exists$t21< t20 state(γ, t21, input(mouse)) |= observed(food_at(X)) & $\forall$t22, t20 $\geq$ t22 > t21 state(γ, t22, input(mouse))|= not(observed(not(food_at(X))))] $\Rightarrow$ $\exists$t23, t23>t20 state(γ, t23, output(mouse)) |= performing_action(goto(X)), where X$\in$ {p1, p2}

BP9: Motivation-based behavior of the mouse (start at position p1)

If the mouse observes no screen and it is not satisfied, and at some time point in the past it observed food at position p1 and since then did not observe food at position p2, then the mouse will go to position p1.

$\forall$t24 [state(γ, t24, input(mouse)) |= observed(not_screen) & state(γ, t24) |= not(mouse_sat) & $\exists$t25, t25<t24 state(γ, t25, input(mouse)) |= observed(food_at(p1)) & $\forall$t26, t26 $\leq$ t24 & t26 > t25 state(γ, t26, input(mouse)) |= not(observed(food_at(p2)))] $\Rightarrow$ $\exists$t27, t27>t24 state(γ, t27, output(mouse)) |= performing_action(goto(p1))

BP10: Motivation-based behavior of the mouse (start at position p2)

If the mouse observes no screen and it is not satisfied, and at some time point in the past it observed food at position p2 and since then did not observe food at position p1, then the mouse will go to position p2.

$\forall$t24 [state(γ, t24, input(mouse)) |= observed(not_screen) & state(γ, t24) |= not(mouse_sat) & $\exists$t25, t25<t24 state(γ, t25, input(mouse)) |= observed(food_at(p2)) & $\forall$t26, t26 $\leq$ t24 & t26 > t25 state(γ, t26, input(mouse)) |= not(observed(food_at(p1)))] $\Rightarrow$ $\exists$t27, t27>t24 state(γ, t27, output(mouse)) |= performing_action(goto(p2))

BP11: Motivation-based behavior of the mouse (continue at position p2)

If the mouse is at position p1 and there is no food at p1 and the mouse is still not satisfied, then it will go to position p2 to continue its search for food

$\forall$t28 state(γ, t28) |= mouse_at(p1) & not(food_at(p1)) & not(mouse_sat) $\Rightarrow$ $\exists$t29, t29>t28 state(γ, t29, output(mouse)) |= performing_action(goto(p2))

BP12: Motivation-based behavior of the mouse (continue at position p1)

If the mouse is at position p2 and there is no food at p2 and the mouse is still not satisfied, then it will go to position p1 to continue its search for food

$\forall$t30 state(γ, t30) |= mouse_at(p2) & not(food_at(p2)) & not(mouse_sat) $\Rightarrow$ $\exists$t31, t31>t30 state(γ, t31, output(mouse)) |= performing_action(goto(p1))

The external behavioral specification for *the delayed-response case of behavior* in the second experimental setting consists of the properties EP2-EP4, BP1-BP8 and property EP5 specified below.

EP5: At some time point food had been put at the position p1, after some time one cup had been placed upon food and another cup had been placed at the position p2; thereafter food has been taken away from p1 and has been put at p2 behind the cup, after that the screen is raised

$\exists$t32, t33, t34, t35, t33>t32 & t33<t34 & t35>t34 state(γ, t33) |= [cup_at(p1) & cup_at(p2)] & state(γ, t32) |= food_at(p1) & state(γ, t34) |= [food_taken_away_from(p1) & food_at(p2)] & state(γ, t35) |= not_screen

The external behavioral specification for *the case of motivation-based behavior* in the second experimental setting consists of the properties EP2-EP5, BP1-BP7, and BP9-BP12.

Consider the case when the mouse has the delayed-response type of behavior. It is described by a property expressing that the mouse goes to the position with food if and only if it observes that there is no screen and at some point in the past the mouse observed food and since then did not observe the absence of food. By the example of this property let us demonstrate how the external behavioral specification of

the agent is transformed into the executable internal behavioral specification. The complete description of the finite state transition system for the first experiment is given below:

present_time(t) ∧ not(∃t13, t13<t ∧ memory(t13, observed(food_at(p1))) ∧ [∀t14, t13 < t14≤ t ⇒ not(memory(t14, observed(not(food_at(p1)))))]) → present_time(t+1)

present_time(t) ∧ not(∃t131, t131<t ∧ memory(t131, observed(food_at(p2))) ∧ [∀t141, t131 < t141≤ t ⇒ not_memory(t141, observed(not(food_at(p2))))]) → present_time(t+1)

present_time(t) ∧ qcprep8 ∧ not(∃t12, t12≥t ∧ memory(t12, observed(not_screen))) → present_time(t+1)

present_time(t) ∧ qcprep9 ∧ not(∃t121, t121≥t ∧ memory(t121, observed(not_screen))) → present_time(t+1)

present_time(t) ∧ preparation(goto(p1)) → present_time(t+1)

present_time(t) ∧ preparation(goto(p2)) → present_time(t+1)

present_time(t) ∧ observed(food_at(p1)) ∧ not(cup_at(p1)) → memory(t, observed(food_at(p1)))

present_time(t) ∧ observed(food_at(p2)) ∧ not(cup_at(p2)) → memory(t, observed(food_at(p2)))

present_time(t) ∧ observed(not(food_at(p1))) ∧ not(cup_at(p1)) → memory(t, observed(not(food_at(p1))))

present_time(t) ∧ observed(not(food_at(p2))) ∧ not(cup_at(p2)) → memory(t, observed(not(food_at(p2))))

present_time(t) ∧ observed(food_taken_away_from(p1)) ∧ not(cup_at(p1)) → memory(t, observed(food_taken_away_from(p1)))

present_time(t) ∧ observed(food_taken_away_from(p2)) ∧ not(cup_at(p2)) → memory(t, observed(food_taken_away_from(p2)))

present_time(t) ∧ observed(screen) → memory(t, observed(screen))

present_time(t) ∧ observed(not_screen) → memory(t, observed(not_screen))

memory(t, observed(food_at(p1))) → memory(t, observed(food_at(p1)))

memory(t, observed(food_at(p2))) → memory(t, observed(food_at(p2)))

memory(t, observed(not(food_at(p1)))) → memory(t, observed(not(food_at(p1))))

memory(t, observed(not(food_at(p2)))) → memory(t, observed(not(food_at(p2))))

memory(t, observed(food_taken_away_from(p1))) → memory(t, observed(food_taken_away_from(p1)))

memory(t, observed(food_taken_away_from(p2))) → memory(t, observed(food_taken_away_from(p2)))

memory(t, observed(screen)) → memory(t, observed(screen))

memory(t, observed(not_screen)) → memory(t, observed(not_screen))

food_at(p1) ∧ not(mouse_sat) ∧ not(food_taken_away_from(p1))→ food_at(p1)

food_at(p2) ∧ not(mouse_sat) ∧ not(food_taken_away_from(p2))→ food_at(p2)

not_screen → not_screen

cup_at(p1) → cup_at(p1)

cup_at(p2) → cup_at(p2)

performing_action(goto(p1)) → mouse_at(p1)

performing_action(goto(p2)) → mouse_at(p2)

mouse_at(p1) ∧ food_at(p1) → mouse_sat

mouse_at(p2) ∧ food_at(p2) → mouse_sat

mouse_sat ∧ mouse_at(p1)→ not(food_at(p1))

mouse_sat ∧ mouse_at(p2)→ not(food_at(p2))

mouse_at(p1) ∧ food_at(p1) → mouse_at(p1)

mouse_at(p2) ∧ food_at(p2) → mouse_at(p2)

present_time(t) ∧ ∃t13, t13<t ∧ memory(t13, observed(food_at(p1))) ∧ [∀t14, t13 < t14≤ t ⇒ not(memory(t14, observed(not(food_at(p1)))))] → qcprep8

present_time(t) ∧ ∃t131, t131<t ∧ memory(t131, observed(food_at(p2))) ∧ [∀t141, t131 < t141≤ t ⇒ not(memory(t141, observed(not(food_at(p2)))))] → qcprep9

present_time(t) ∧ qcprep8 ∧ ∃t12, t12≥t ∧ memory(t12, observed(not_screen)) → preparation(goto(p1))

present_time(t) ∧ qcprep9 ∧ ∃t121, t121≥t ∧ memory(t121, observed(not_screen)) → preparation(goto(p2))

preparation(goto(p1)) → performing_action(goto(p1))

preparation(goto(p2)) → performing_action(goto(p2))

Using the state transition system representation simulations of the agent internal dynamics and verification of the agent model with respect to a general system property can be performed. As has been mentioned before, the general property for the both experiments represents a satisfactory condition, namely for all traces if the screen is removed and food is hidden under the cup at the position p1, then the mouse will be eventually satisfied. Or in CTL,

AG (not_screen & food_at(p1) & cup_at(p1) → AF mouse_sat) (7)

where ***A*** is a path quantifier defined in CTL, meaning “for all computational paths”, ***G*** and ***F*** are temporal quantifiers that correspond to “globally” and “eventually” respectively.

In order to perform verification by means of SMV model checker, the general description of the finite state transition system has been automatically translated into the SMV model specification format. The automatic verification showed that the property (7) satisfies the model of the agent delayed-response behavior.

Now, consider the case of the motivation-based behavior of the animal. It is described by a property which expresses that if the screen is removed and the mouse is not satisfied (has hunger) and it observed a position of food before, then it will start searching for food first at the position of last observation and then, if it can not find food there, it will continue searching at the other position. Such specification of behavior can be attributed for example to an animal that feels strong hunger. The model of agent behavior for this case also satisfies the general property (7).

In the second experimental setting the mouse observed food for some time at the position p1, after that one cup is put covering the food and another cup is put at the position p2. Thereafter, invisibly for the mouse food is removed from position p1 and put under the cup at position p2. Later the screen is raised and the animal is free to go to any position.

The global property for verifying expresses that for all traces if the screen is removed and food is hidden behind the cup at the position p2, then the mouse will be eventually satisfied.

AG (not_screen & food_at(p2) & cup_at(p2) → AF mouse_sat) (8)

The automatic verification in SMV showed that the model of the agent behavior for the delayed-response case does not satisfy the property (8). From the counter-example generated by the model checker it is visible that the animal went to the position p1, and did not find food

there, which caused the failure of the satisfaction property. As has been confirmed by automatic verification, the external behavioral specification for the case of motivation-based behavior satisfies the property (8).

From the results of verification of external behavioral specifications for both cases of behavior in both experimental settings with respect to the satisfaction properties (7) and (8) we draw the conclusion that the mouse that manifests motivation-based behavior fits more for surviving in the world, described by the experimental conditions than the mouse that has the delayed-response behavior.

REFERENCES

[1] Fitting, M. First-order Logic and Automated Theorem Proving. 2nd edition, Springer-Verlag, 1996.

[2] McMillan, K. Symbolic Model Checking. Kluwer Academic Publishers, 1993.